Transfert Learning & GAN

By Guillaume Thomas and Pejman Samieyan

1 - Transfert Learning through feature extraction from a CNN

1.1 - VGG16 Architecture

1) Knowing that the fully-connected layers account for the majority of the parameters in a model, give an estimate on the number of parameters of VGG16 (using the sizes given in Figure 1).

Here is a table summarizing the last layers involving weights with a fully-connect layer:

Layer	Tensor dimension	Number of neurons
Max pooling 5	$7 \times 7 \times 512$	$25\,088$
Fully-connected 1	$1 \times 1 \times 4096$	4096
Fully-connected 2	$1 \times 1 \times 4096$	4096
Fully-connected 3	$1 \times 1 \times 1000$	1000

Hence, we can approximate the total number of weights of VGG16 with:

$$25\,088 \times 4\,096 + 4\,096 \times 4\,096 + 4\,096 \times 1\,000 = 123\,633\,664$$

2) What is the output size of the last layer of VGG16? What does it correspond to?

The output layer is composed of 1000 neurons. This number corresponds to the number of classes this architecture is trained on.

1.2 - Transfer Learning with VGG16 on 15 Scene

5) Why not directly train VGG16 on 15 Scene?

There are two reasons a simple training of VGG16 on 15 scene dataset won't work:

- First, the number of classes on the output layer don't match between the two datasets ($1\,000$ for ImageNet, against 15 for 15 scene). Hence, the VGG16 architecture needs to change.
- Even if the previous point was not an issue, a full re-training will loose the previously learned weights. To solve this issue, a fine tuning strategy is often applied. This strategy consists of blocking the backpropagation of gradients in the pre-trained network (or a part of it) or by using a diminished gradient. These strategies help to preserve the weights of the pre-trained network.

6) How can pre-training on ImageNet help classification for 15 Scene?

Thanks to its pre-training on ImageNet, VGG can now extract some meaningful information (e.g. lines, some particular shapes like cercles or even chairs). This knowledge will ease the learning by diminishing the amount of work necessary to learn the structure of inputted images. Hence, it diminishes the need for data (i.e. amount of images in this context) on the target task and reduces time and ressources required to train such a model.
This last point is important as it opens possibilities of experimentation to a broader audience. This way, students and researchers that do not have access to GPU clusters can still successfully train and use large models. This woud not be possible otherwise, in particular with the trend for large neural networks.

7) What limits can you see with feature extraction?

First, extracted features are intrinsecally harder to interpret for a human, compared to raw features (which usually have a meaning, like the surface of a house, or the gray value of a pixel). This lack of interpretability can hamper debugging or model improvement.
Then, to perform well, the feature extraction need to be undergone in similar source and target tasks. Indeed, we can expect a model trained for shape classification to perform painfully if inputted with features extracted from a color classification model. This need for correlation between the two tasks can be split more precisely:

- The extracted features should be as descriptive as possible, in order to be used smoothly in the target task.
- The two tasks should share common invariance need (like translation or radiometric invariance).

8) What is the impact of the layer at which the features are extracted?

The layer at which the features are extracted has some importance for the following reasons:

- The depth of layer influence the abstraction level of the features. Shallow layers will extract simple low-level (like lines). On the other hand, deeper layers will extract more complex high-level features (like the presence of a wheel).
- The layer ouput tensor dimensions define the size of the extracted features. Note that with Convolutional Neural Network this tends to be less true. Indeed, the dimensionality reduction by maxpooling layers on feature maps are usually compensated by a higher number of output filters.
- The depth of the layer chosen influence the computational cost of the feature extraction. This choice can be significant on embedded devices.
- Lastly, the layer cut defined if the deep feature extractor can handle any image input size (i.e. if no fully-connected layer is kept).

9) The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?

There are several solutions that can be applied to solve this issue:

- The most naive one is to copy the black-and-white channel in the three R, G, & B input channels.
- One might try to add a convolution layer with 3 filters before the input of the neural network. After a fine-tuning, this convolutional layer might be able to recover 3 channels tensor suitable for the RGB input layer.
- Another approach is to modify the model. Indeed, we can sum the weights of the filters from the first layer along the RGB dimensions. This way we recover the features computed on the image from the first approach. This approach do save some computation work for the first layer.
- Lastly, GAN models for black-and-white-to-color might also be a promising solution.

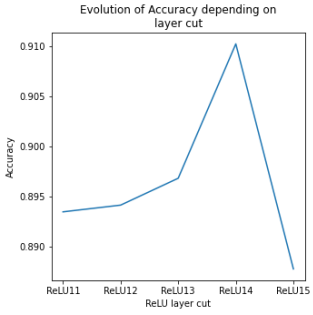
10) Rather than training an independent classifier, is it possible to just use the neural network? Explain.

As stated in question 5, the model can't be used as is. Though, after a small modification on the last layer of the model (e.g. by adding several fully-connected layers), the model can be trained.
A simple training might not guarantee sweet performances, especially under a small dataset. A better suite approach is to fine-tune it. This strategy consists of blocking the backpropagation of gradients in the pre-trained network (or a part of it) or by using a diminished gradient. These strategies help to preserve the weights of the pre-trained network.

11) For every improvement that you test, explain your reasoning and comment on the obtained results.

Impact of the depth of the feature extraction

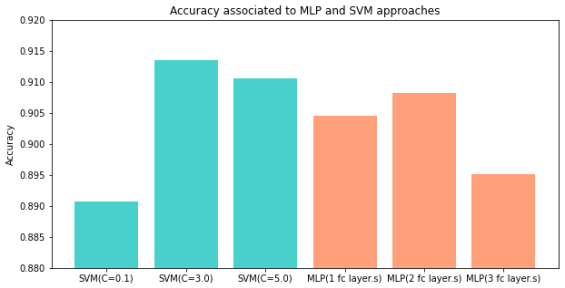
As discussed before in question 8, the depth of feature extraction has some influence on a variety of factors. In the following graph[1], we observe its meaningful influence on the accuracy. Indeed, the best layer cut offers a 9.0% missclassification rate, against a 11.2% for the worst one. This is a 24% relative difference.



We can extract two main patterns from this graph. In a first phase, until the "ReLU14" layer, the accuracy seems to increase. This can be explain by higher level of abstraction for deeper layer. This abstraction seems to help the learning of the SVM. Nevertheless, the last layer cut, at "ReLU15", sees a significant dropout in performance. This phenomenon might be explained by an over-nesting of this layer abstration with the Imagenet source task.

SVM and MLP comparison

Finally, we tuned the parameter C of the SVM approach, and compared it with a MLP head directly connected on the feature extractor[2]. Below is a bar plot summarizing our findings.

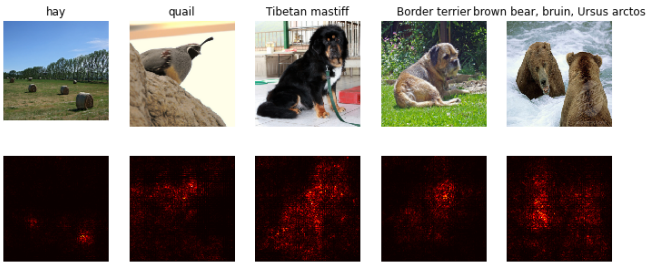


In the end, it seems that both Support Vector Machine (SVM) and Multi Layer Perceptron (MLP) have similar performance, when well-tuned. Here, they both reach a pick performance of 91%.

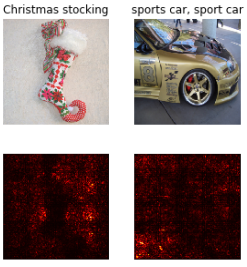
2 - Visualizing Neural Networks

2.1 - Saliency Map

1) Show and interpret the obtained results



On the image above, we can observe the result of the saliency map computation on a few example images. Interestingly, *Squeezenet* uses different clues in each image to deduce its class. In that respect, it is using the whole object to detect “Tibetan mastiff” and “hay”. But for other classes, it seems to only use a part of it, namely the head in the case of the “Border terrier” above.
Lastly, in the example of “Christmas stocking” and the “sport car”, the object had actually no or little influence in the classification. It seems that *Squeezenet* lack of generalization for those two examples. Their saliency maps indicate that the classification decision was, repectively, based on the white wall and the concrete beneath the sport car. It might be due to a smaller set of highly correlated samples for these two datasets.

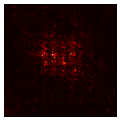


From my point of view, this approach seems to provide great insights for model interpretability. We can deduce evidence of a poor learning (as in the example of the “Christmas stocking” and the “sport car”). The other way around, we can observe distinctive properties of some class (like the face of the “Border terrier”), that enable their efficient classification.

2) Discuss the limits of this technique of visualizing the impact of different pixels.

This approach has some main downfalls, though:

- First, the saliency maps are really noisy. This noise comes several sources, including max pooling layers. With their presence, like in *SqueezeNet*, gradient becomes sparse, because of the max operation. This results in a “grid-like effect”, which increases the noise in the saliency map. This phenomena is especially visible in the following saliency map from a toilet paper image.



- Secondly, the qualitative aspect of saliency maps makes it a poor judge of the correctness of our network. It is in fact hard to anticipate the network classification only based on the saliency map. Indeed, with the following example, *SqueezeNet* seems to focus its attention on the right part of the image, but failed to recognize toilet paper and actually predicts a pillow.

  This hampers our hability to understand why the network is failing, which is one of the main purpose of Saliency map.

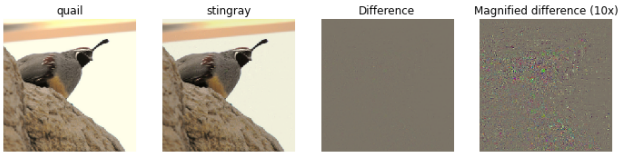
- Pixel attribution methods can be very fragile. Ghorbani et al. (2019) showed that introducing small (adversarial) perturbations to an image, which still lead to the same prediction, can lead to very different pixels being highlighted as explanations.

3) Can this technique be used for a different purpose than interpreting the network?

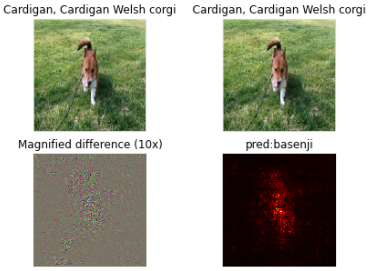
We could imagine to use the saliency map of the model as a zero-shot rough segmentation of the image. It can already produce great results, as for the example of the “Tibetan mastiff” above.

2.2 - Adversarial Examples

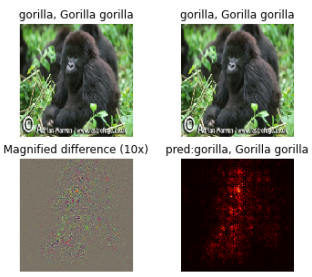
5) Show and interpret the obtained results.



Generated adversarial examples, such the one above are indifferent from their original images from the human eyes. Even the difference map without magnification seems flat.
Interestingly, it appears that a correlation exists between the main modified regions of adversarial examples and the most activated regions within saliency maps. This correlation does make sense. Indeed, if one wants to fool the network, it is of its interest to modify the regions on which the network makes its decision on.
We can observe this phenomena in the following image with the adversarial example and its magnified difference on the left and the original image and the saliency map of *SqueezeNet* on the right.



And here is another example of the same phenomena with a gorilla.



Lastly, we can observe that all images do not require the amount of modification to fool the network. In the example below, the modification are truly undetectable, though the network is fooled. We might think that this tendency illustrate a smaller discrimination capacity of the network between this pair of classes. The

network might have lack of learning to discriminate them.

6) In practice, what consequences can this method have when using convolutional neural networks?

In practice, the ability of this attack to produce fooling examples for a given class can be malicious, especially in context like federated learning. Indeed, with access to the model, any malicious users could produce hard-to-detect fake examples and try to corrupt the federated model.

2.3 - Class Visualization

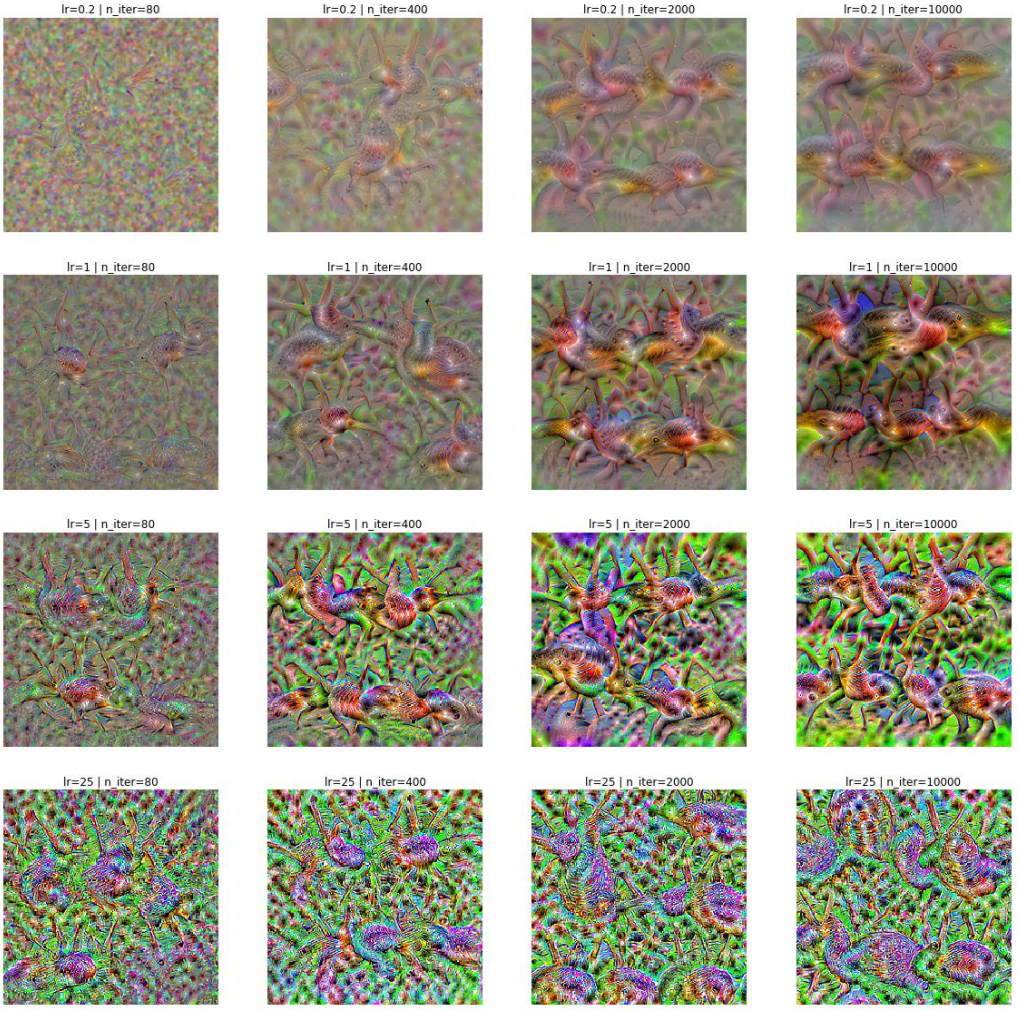
8) Show and interpret the obtained results.

The class visualization brings up meaningful patterns, like the fur of the Gorilla, the glass and its reflect of the hourglass, and the slimy aspect of the snail. Even though, the generated image is far from realistic DALL-E's images, it still successfully illustrate key features *SqueezeNet* is focusing on in its classification task.

More particularly, this approach succesfully recovers the colors, the aspect, the shades and a part of the global shape of the targeted object. We can imagine that the more complex and realistic a Class Visualization is, the better is the learning of the associated network.

9) Try to vary the number of iterations and the learning rate as well as the regularization weight.

We conducted an experiment to measure the impact of the learning rate and the number of iterations on the resulting class visualization. Here are the results of our experimentation:

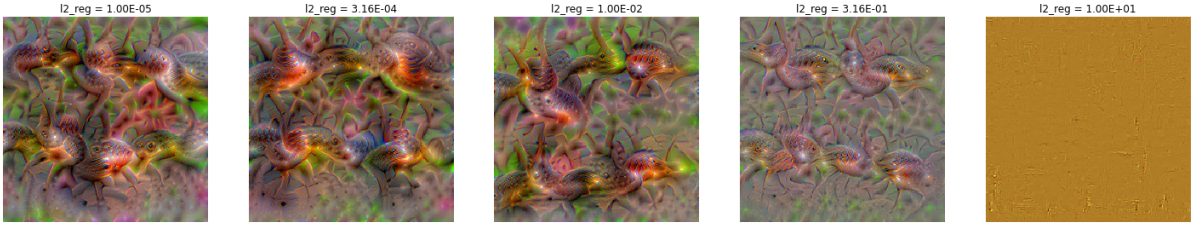


First, a minimum threshold of learning must be reach to obtain colorful image. Otherwise images remain dull. This threshold can be reach either by increasing the learning rate or the number of iterations. Therefore, the most dull images of our experiment are located in the top-left corner of the previous visualization, where the learning rate and the number of iterations are both low.

This experimentation showed that with a learning rate above 5, we can observe that the images start to take more vivide colors and, at an extrem, becomes noisy. These learning rates are too high to find a global minima.

The best results in our experimentation were obtained with a learning rate of 1 and $10\,000$ iterations.

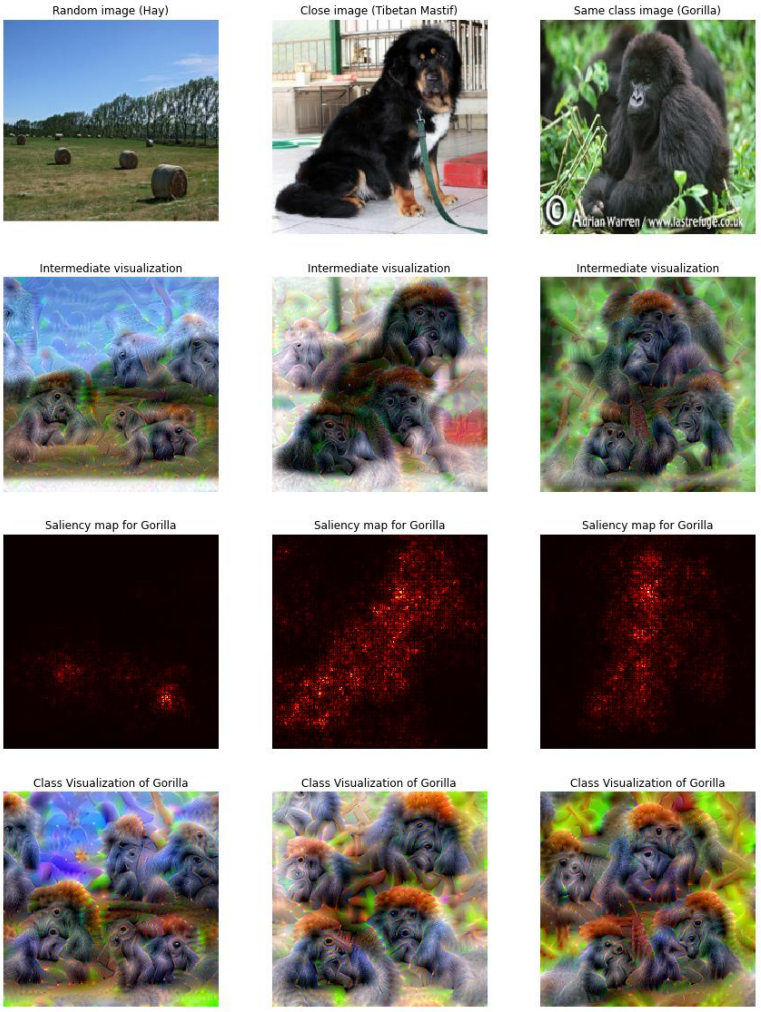
We also conducted a second experiment to measure the impact of the L2 normalisation on the result. Here are our findings:



Between $[10^-5, 10^{-2}]$, the regularization does not seem to have significant influence on the outputed image, even if the order of magnitude varies greatly. Though, this parameter still has an important role, otherwise, with higher values, color starts to disapear and even the shapes eventually vanish. v

10) Try to use an image from ImageNet as the source image instead of a random image (parameter `init_img`). You can use the real class as the target class. Comment on the interest of doing this.

Class Visualization method can be initialized with any images, and depending on this one, different behaviors emerge. Here is below the result of our experiment:



On the first row are presented the input images. On the second row, we can observe the intermediate result after 500 iterations of the class visualization. This intermediate state illustrates region from which the patterns are emerging. We can see below, with the saliency maps, that these regions are the one detected by *SqueezeNet* as the most probable region containing a Gorilla.
Lastly, the background color of the class visualization seems to be inherited from the original image. Indeed, the hay image with a great blue sky produces a class visualization with a blue background.

3 - Domain Adaptation

1) If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen?

Without the GRL component, the network will perform horribly. Indeed, in this context, the feature extractor will be optimized on the domain classification task, rather than on the domain confusion. This will encourage the learning to produce features easily separable for each domain. Hence, the target dataset won't benefit from the supervised learning on the source dataset and will perform horibly.

2) Why does the performance on the source dataset may degrade a bit?

The need for domain confusion changes the feature extractor. Those changes can partially be absorbed by the classifier head. But the rest of the unabsorbed modifications hence negatively impact the performance on the source dataset.

3) Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.

The factor used in the GRL is a weight for the optimization of the domain confusion. With a factor of 0 at the beginning of the training, the gradients are then nullify when backpropagating through the GLR. Those nullify weights won't, therefore, optimize any parameters. This factor, hence, allows the training to first focus on the supervised training of mnist classification and then progressively optimize jointly for domain confusion. This transition of training task is possible thanks to the factor schedule.

4) Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.

In manner similar to UDA, Pseudo-labeling is another semi-supervised learning approach. It combines supervised and unsupervised data to learn. Pseudo labeling is undergone in several steps and improved iteratively.
A first model is trained on the supervised dataset. This model is used to predict “pseudo-labels” from the unsupervised dataset. The most confident ones are then kept and added to the training set. The model can then be retrained and sees its results improved.
This method can be applied to a variety of models. Intuitively, it reduces the overlap of class probability distribution. Therefore, it favors a low-density separation between classes, which helps classification.

4 - Generative Adversarial Networks

4.1 - Generative Adversarial Networks

1) Interpret the equations (6) and (7). What would happen if we only used one of the two?

The first equation’s goal is to make the generator produce data that the discriminator will classify incorrectly. The second equation aims to improve the discriminator’s ability to correctly classify real data and decrease its error rate when classifying generated data. Without using both equations, either the generator or discriminator would dominate and produce subpar results.

2) Ideally, what should the generator G transform the distribution P (z) to?

Ideally, the generator G should change the distribution P(z) to match the distribution P(X) of the real data.

3) Remark that the equation (6) is not directly derived from the equation 5. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the “true” equation be here?

The “true” equation should be:

$$\min_G E_{z\sim P(z)}[log(1 - D(G(z)))]$$

4) Comment on the training of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.

The model generates images with the desired shape, but the quality is low due to low resolution and blurriness. Despite this, early results have been promising and the shapes are clearly visible. However, it’s important to note that both classifiers have unstable loss.

5) Comment on the diverse experiences that you have performed with the suggestions above. In particular, comment on the stability on training, the losses, the diversity of generated images, etc.

Reducing the size of the latent space nz to 10 does not result in a significant change in the visual outcome of the images, likely due to the low resolution of the images. This may also indicate that the results are simpler as it would be challenging to model the data with such a small vector. On the other hand, using a larger vector size such as 1000 would increase the computation time for mapping the vectors to the data.

4.2 - Conditional Generative Adversarial Networks

6) Comment on your experiences with the conditional DCGAN.

7) Could we remove the vector y from the input of the discriminator (so having cD(x) instead of cD(x, y))?

The vector y inputted to the discriminator is the main peculiarity of Conditional GAN. Without it, the architecture used would simply be summarized to a classical GAN (like DCGAN). Indeed, without the input of y to the discriminator, the generator has no reasons to learn the generation of conditional images. In this context the vector y would simply be considered as noise.

8) Was your training more or less successful than the unconditional case? Why?

The conditionning of GAN is actualy helping the network to learn and, therefore, converges faster. Intuitively, the prior information on class helps the generator to perform domain confusion. Whereas, in classical GAN, without this prior, the generator must learn a mapping between the noise space into the image space that will mimic the real class image distribution. The later is much harder.

9) Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here?

1. Note that, we did not experiment with fewer layers as a feature extractor. This is due to two reasons:
    - First, features extracted on shallow layers have larger representation size, and thus do not fit in memory anymore. Indeed, the SVM training needs the full dataset loaded in memory, on the opposite of neural networks which are usually trained by batch. Extracted features of the first layers (i.e. any layers before the first pooling) have the following dimensionality:

      $$num\_filters \times image\_width \times image\_height = 64 \times 224 \times 224 \approx 3.2 \times 10^6$$

      In the end, this results in the following needed RAM to be stored:

      $$3.2 \times 10^6 \; features \times 1\,500 \; samples \times 8 \; bytes \; per \; float = 38\text{GB}$$

      38GB is unfortunately too significant to be stored on any conventional laptop or computer. In the same way, it cannot be stored on the free-tier Google Colab service.
    - In the end, the limitation discussed in the previous point does not matter that much. Indeed, a feature with such a large representation size will hamper training, especially compared to the limited dataset size (only $1\,500$ samples).

    ↩

2. Only fine-tuning with a blocking gradient on the feature extraction have been tried. ↩